VegAu, a software to test a theoretical self-reliance with organic products and a plant-based lifestyle

– OPERATION MANUAL –

– 2017 –

Author on:

ResearchGate : Emy Lee

GitHub : Iscobia

GitHub repository : https://github.com/Iscobia/VegAu

Table of Contents

Index of Tables

Table of Figures

1. VegAu's structure: files organization

The LibreOffice and Excel tables 'inputFR.ods' and 'inputFR.xlsx' show the content of the database that has been imported in the program. The 'ENVIRONMENT' sheet refers to environmental data from MétéoFrance (climate) and from the INRA (soil data from the BDAT and BDETM). 'PLANTS' refers to the compiled data detailed in the chapter 3.2 of the Master thesis, and 'NUTRITION' to the Ciqual one. Pink values of the 'PLANTS' sheet indicate estimations and data that have been emphasized in bold and red in the 'ENVIRONMENT' sheet have been computed by interpolation.. A partially labelled list details the data origin in the 'Sources' sheet.

'Functions_step1.py' and 'Functions_step2.py' contain the secondary functions used for the steps 1 and 2. The latter are then imported in 'VegAu.py' and are reused in the primary functions. As the step 3 only contains two functions that are far not as complex as the ones from the both previous steps, this part does not have a primary function in 'VegAu.py'. Another reason for keeping `MDL_QTTperPerson` out from 'VegAu.py' is that this function imports the data from 'CanadaHealth.py': importing such a big database for only one, relatively short function may slow down the whole program. That way, all functions related to dietary informations stay in 'Functions_step3.py'.

After importing these three modules, 'VegAu.py' runs successively its three main functions: MDL_eligibilityTest, MDL_Rotation and MDL_QTTperPerson. The results are saved in three tables:

- 'dietary_results.csv' to evaluate the feasibility of the computed rotations at the scale of the country;
- 'results_Mapping.csv' to easier synthesize the results in a mapping software like QGIS;
- 'results_RotationAnalysis.csv' to get an overview of the computed rotations for each spatial unit (duration, crops amount, crop type, limiting factors...).

If interrogations remain after the next chapter, each function from these python files are fully commented within the code for further information.

All following variables as well as their main particularities are summarized in the Appendix 1.

2. Functions classification

Functions have been classified in 7 categories according to their prefix in order to clarify their role in the model (see Table 1).

Another classification in 4 categories relies on their position in the program's hierarchy: tertiary functions are parts of secondary ones and secondary functions are used in primary ones. The latter, also called "main functions" in this work, are though the more complex ones. The last hierarchical category is the "variable function", mainly lambdas.

Lambdas often only return the data from the 'plants' and 'environment' dictionaries: they are only used to simplify the code and make it easier to read. For instance, GSmin(crop) return the minimum Growing Season length for the given crop according to the dict 'plants' and PRAsurface(PRA) the agricultural surface of the given PRA according to the dict 'environment'.

Prefix	Signification	Hierarchical position	Comment
MDL	Modelling	Primary function	MDL functions are the 3 main VegAu's function
ASSESS	Assessment	Secondary or tertiary function	The ASSESS functions are the most numerous ones. Their outputs are either updated crop selections or selection indices, or both.
SELECT	Selection	Secondary function	The only 2 SELECT functions follow the ASSESS ones: they select a main crop and a cover crop according to their selection indices.
APPLY	Applying	Secondary function	APPLY functions apply the environmental impact of selected crops to the environment (harvesting, residues decomposition)
UPDATE	Update	Tertiary function	The only UPDATE function updates the pests and diseases related variables.
VERIF	Verification	Tertiary function	The only VERIF function plays a great role in the detection of the limiting factor by verifying which crops are still eligible or not while the nutrients assessment (step 2).
None		"Variable function"	They are often lambda functions which allow to get a clearer code by associating directly the variable ID to the related element (crop or spatial area). They can also be more complex functions, but they always return a numerical value.

Table 1: Functions classification according to their prefix - Prefix signification and hierarchical position in the model.

All functions as well as their main variables and secondary functions are summarized in the Appendix 4.

3. Technical overview

3.1. First step: eligibility assessment for each crop

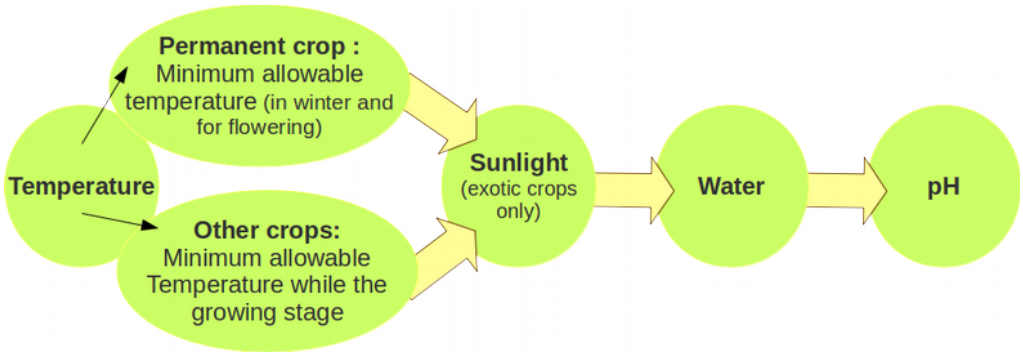

Figure 1: Schematic representation of VegAu's step 1. For each spatial unit, each crop from the database get tested to verify if its requirements match with the local environment (climate and soil).

The selection of eligible crop occurs in the function `MDL_eligibilityTest`. For each PRA, each crop of

the 'plants' database is subject to an eligibility test to verify its minimum supported temperature, water requirement and preferential pH window match with the climate and soil properties of the PRA. These tests occur with the functions `ASSESS_Tmin_germ_forFruits`, `ASSESS_Tmin`, `ASSESS_Water`, and `ASSESS_pH`, fulfilling the list `eligible` with `False` or `True` depending on the eligibility or ineligibility of the crop, except for `ASSESS_Tmin_germ_forFruits`, which returns a `True` or `False` value that is directly used in the function `ASSESS_PRAeligibility`. This next function uses the list `eligible` and select from this list only crops for which all eligibility tests returned `True`. Finally, the remaining crops are copied in the sheet “PLANTS” as lists of IDs and name in English, French and German. These name lists allow the model’s users to see easily which crop was selected for which spatial area. It will be especially useful for further updates to display properly the results according to the language of the interface.

Once a crops selection have been assessed for each PRA, several indexes are calculated for each crop from the ‘plants’ database in order to help the crops choice in rotations (see Table 2). These indexes are:

- `ratioADAPT`: the “adaptability ratio” which corresponds to the sum of the agricultural surface of all PRA for which the crop is "eligible" divided by the total agricultural surface in France.
- `PRIORITYgeneral`: this index is common to all crops. It classifies the crops according to their adaptive capacity: the lower the adaptive capacity, the higher the priority.
- `PRIORITYfruits`: this priority index only concerns fruit trees and berries. Crops are classified from 1 (highest priority) to 5 (lowest priority) according to their potential weekly yield per person.
- `PRIORITYfibre`: it only concerns fibre crops. Because cotton is the most demanding crop and because it’s fibre is the most popular one, it get the highest priority. As it is originally grown under subtropical climates, its low adaptability ratio in temperate ones should be enough to select it among other crops while computing a rotation. However, it seems important to give it priority if all factors are gathered for it to grow. The second highest priority is given to fibre flax for the delicacy of it fibre, just like cotton. Other fibre crops get all the same, lower priority.

Priority index	`PRIORITYgeneral`	`PRIORITYfruits`		`PRIORITYfibre`
	Adaptability ratio	Weekly quantity per person (QttPerInhabitant, kg of fruits)		Crop type (only for fibre crops)
		Fruit trees	Berries	
1 (highest priority)	Between 60% and 80%	Less than 1	Less than 0,1	Cotton
2	More than 80%	From 1 to 2	From 0,1 to 0,2	Fibre flax
3	/	From 2 to 3	From 0,2 to 0,3	Other fibre crops
4	/	From 3 to 4	From 0,3 to 0,4	/
5 (lowest priority)	/	More than 4	More than 0,4	/

Table 2: Assignment of priority indices

3.2. Second step: assessing a rotation for each PRA

The function `MDL_Rotation` creates an optimal rotation using crops which have previously been selected as eligible. This function tests each crop from the 'plants' database according to the environmental data (‘environment’ database).

The dictionary x.rotat is a dictionary containing lists of tuples for each spatial unit. These tuples are updated at several steps of the rotation's simulation and are composed that way:

```
(x.SelectedCrop, x.SelectedCC, x.rotat[PRA][-1][2], x.EndPreviousCrop_later)
```

For more informations about this variables and their related functions, see Appendix 2.

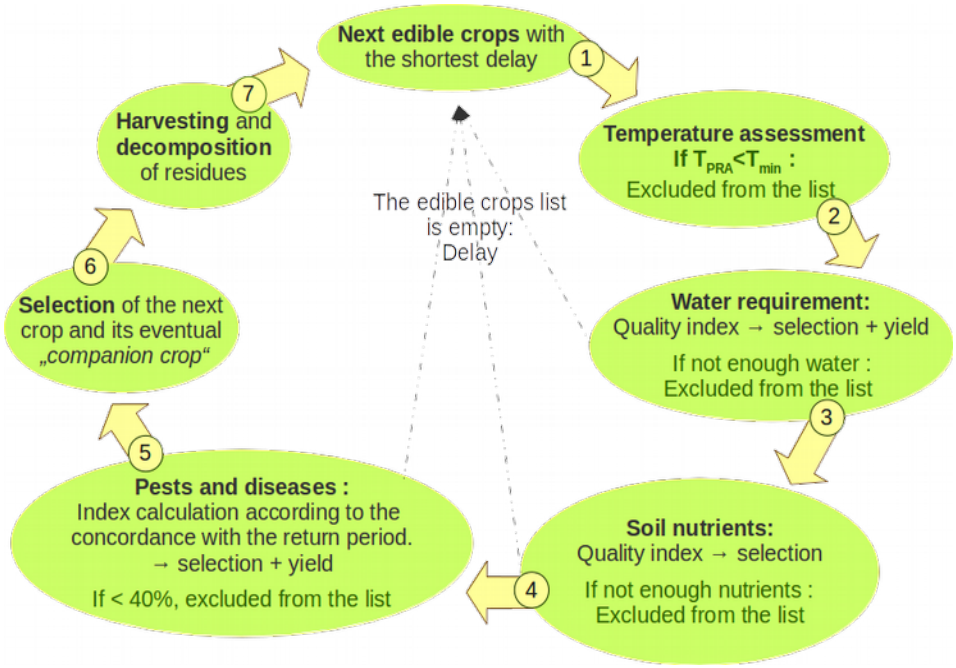

Figure 2: Schematic representation of VegAu's 'step 2'. For each spatial unit (in this case "Petites Régions Agricoles" from France), requirements of each eligible crop get tested to compare them with the local conditions. For each test, an index is calculated. The four indices are then summarized in a single selection index to chose the crop with the better characteristics. The calculation continues until there is not enough nutrients any more in the soil.

3.2.1. ASSESS_SeedingDate(PRA, x) : the shortest delay between crops

The first rotation's month is set to March in order to allow the plantation in early Spring (`x.EndPreviousCrop_earlier` and `x.EndPreviousCrop_later` are equal to 3). At the beginning of the rotation, the eligible crops are then either the ones which can be directly planted in March or the earliest next ones. Afterwards, they correspond to those which can be planted between the earliest and the latest harvesting date of the preceding crop, respectively `x.EndPreviousCrop_earlier` and `x.EndPreviousCrop_later`. If the selection only contains three crops or less, the program looks for the both earliest next planting dates and adds the concerned crops to the dictionary `x.laterCrops`. The dictionary `x.indexDelay` associates an index to each crop that has been selected in `x.eligibleCrops` according to the eventual delay between their planting and the previous crop. If there is no delay, the index is equal to 1.

As trees are permanent crops, they are added automatically to the list for the first rotation loop. Permanent crops can only be selected as first crop of the rotation, except if the first one is a cover crop. If that is the case, a permanent crop can be chosen in second position.

3.2.2. ASSESS_WaterResources(PRA,x): Climatic conditions

As the first environmental test refers to **temperature**, this function should be renamed or splitted. It is a very

short but eliminating step: if the temperature falls just once below the minimum temperature while its shortest growing season, the assessed crop is deleted from the list. If there is no eligible crop any more after this step, a delay because of the “cold season” is saved in `x.rotat[PRA]` :

```
('Cold season', None, x.EndPreviousCrop_earlier)
```

Water resources are evaluated just like in the first part of the program thanks to the function `WaterResources(month,GSstart,PRA,crop,x)`. Water requirement is divided in four values to represent the evolution of the plant’s needs while its different growing stages. The quality of water resources is estimated by calculating the following index:

```
1- (water_stress_threshold / WaterResources(month, x.GSstart[crop], PRA, crop, x))
```

If the index becomes negative – if water resources are lower than the water stress threshold – the corresponding crop is deleted from the list. If there is no eligible crop any more after this step, a delay because of the “dry season” is saved in `x.rotat[PRA]` :

```
('Dry season', None, x.EndPreviousCrop_earlier)
```

When such a delay is registered, tests resume to the function `ASSESS_SeedingDate(PRA, x)` to chose later crops.

3.2.3. ASSESS_Nutrients(x, PRA): Soil nutrients availability

If there are still crops in the selection, the program verifies if there are **enough nutrients in the soil** to grow them.

`ASSESS_NutrientsMargin(PRA,x)` takes into account the actual stand of the soil nutrients (`x.ActualStand`) and of the decomposing organic matter (`x.decomposition_month`). Margins are calculated that way :

- if `nutrient` is another nutrient than N : `x.ActualStand[PRA][nutrient] + x.decomposition_month[monthInGS][nutrient] – (removed[nutrient] / GSmin(crop) )`
- if `nutrient` = N : `x.ActualStand[PRA][nutrient] + x.decomposition_month[monthInGS][nutrient] - ((removed[nutrient] - **fixedN(crop)**) / GSmin(crop) )`

`x.decomposition_month` contains the nutrient amount which should be released to the soil after the death of each crop of the rotation. Its keys correspond to the duration (in month) after which the nutrients are released, that is added to the PRA soil resources. For each month of the rotation, the values of each “decomposition month” are switched to the key "month - 1" and the nutrients that was referenced in the key 1 are added to the PRA soil nutrients.

The dictionary `removed` contains the total amount of nutrients that the assessed crop needs to grow, including harvested parts as well as future residues.

`ASSESS_NutrientsMargin(PRA,x)` returns a list with the nutrient margin of each crop from x.eligibleCrops that have **only positive** nutrient margins. If the margin is negative, it means that the nutrient amount is insufficient in the soil to grow the crop. The latter is thus deleted from the list and the corresponding resource is added to the list of limiting factors – `x.LimitingFactors`.

If `x.eligibleCrops` is not empty, an index is calculated in two times in the dictionary `x.indexNutrients`. First, an average value of all nutrient margins is calculated for each crop. Next, all these values are standardized according to their maximum to get an index between 0 and 1.

If `x.eligibleCrops` is empty, `x.LimitingFactorReached` is set to `True` and the entry `('Limiting`

`factor', x.LimitingFactor[PRA], x.EndPreviousCrop_later)` is added to `x.rotat`: the limiting factor is saved and the rotation is over.

3.2.4. ASSESS_PestDiseases(x, PRA): Sensibility to pests and diseases

If `x.eligibleCrops` is not empty, the sensibility to pests and diseases is evaluated in the dictionary `x.indexPnD[crop]`.

The indexes it contains is calculated this way : `(duration_since_previous_crop /12 )/ period(crop)`, where `duration_since_previous_crop` is the calculated as follows :

```
x.VERIFprodBOT[prodBOT(crop)]['Duration since previous crop'] +
                growing_season_of_the_last_crop
```

If `x.indexPnD[crop]` is lower than 1, the pests and diseases risks are considered as being proportional to the index, that is the actual lapse of time between both crops. If the resulting percentage is lower than 40%, the crop is deleted from the list.

If `x.indexPnD[crop]` is greater than or equal to 1, it is assumed that there is no pest or disease risk. If the index is greater than 1, it is reset to 1.

3.2.5. SELECT_CashCrop(x,PRA,data): Selecting the best main crop

If there is less than 120 kg N/ha, VegAu chooses among the eligible cover crops to regenerate the soil (list `eligibleCoverCrops`). Else, several thematic sub-lists are created from the remaining crops from the `x.eligibleCrops` list. The program chooses among them giving priority to the first ones, in the order cited below:

1. `eligible_permanent_crops = [c for c in x.eligibleCrops if prodCAT(c) == 1 or prodCAT(c) == 2]`
2. `unusedCrops_countryScale = [c for c in unusedCrops if c not in x.totalYields["TOTAL"] and c not in delay]`
3. `UnusedCrops = [c for c in unusedCrops if c not in x.totalYields["TOTAL"] and c not in delay]`
4. `unusedCrops_without_delay = [c for c in unusedCrops if c not in delay]`
5. `unusedCrops_countryScale_delay = [c for c in unusedCrops if c not in x.totalYields["TOTAL"]]`
6. `unusedCashCrops = [c for c in eligibleCashCrops if c in unusedCrops]`

If the `x.eligibleCrops` or one of these lists only contains one entry, this crop is directly chosen to continue the rotation. Else, a final selection index is calculated :

```
Final_Eligibility_Index[crop] = round(((x.indexDelay[crop] + x.indexWR[crop] + 0.5 *
                                x.indexNutrients[crop] + 2 * x.indexPnD[crop]) / 4.5), 2)
```

`x.indexDelay[crop]` and `x.indexWR[crop]` are weighted by 1, `x.indexNutrients[crop]` by 0.5 and `x.indexPnD[crop]` by 2.

After the selection of the crop with the greatest index, `UPDATE_VERIFprodBOT_and_PestsDiseases_in_rotation(PRA, x)` creates an entry in `x.VERIFprodBOT` for the newly selected crop if there is no one in the dictionary. Then, it verifies if the minimum return period is respected : if it is respected, `x.VERIFprodBOT[prodBOT(x.SelectedCrop)]['Duration since previous`

crop'] is reset to zero and the dictionary x.PestsDiseases_in_rotation[PRA] remains untouched. Else, x.PestsDiseases_in_rotation[PRA] is incremented with 1. This value is never used as a variable in the program, it is only an indicator for the user to easily compare several rotations.

Then, x.GSstart takes the value of x.GSstart[x.SelectedCrop] and becomes a float. Following variables are also updated :

- x.EndPreviousCrop_earlier = int(x.GSstart + GSmin(x.SelectedCrop))
- x.EndPreviousCrop_later = int(x.GSstart + Gsmax(x.SelectedCrop))
- YIELD = (expYIELD(x.SelectedCrop) * PRAsurface(PRA))* x.indexWR[x.SelectedCrop] * x.indexPnD[x.SelectedCrop]

Finally, YIELD is rounded to the nearest thousandth and added to x.totalYields[PRA][x.SelectedCrop].

3.2.6. SELECT_CompanionCrop(x, PRA): Choosing a companion crop

Once a crop is selected, the model verifies if there is enough water and nutrients to grow a companion crop with the main one without penalizing the latter.

Just like for the main crop, the function ASSESS_Water_CompanionCrop (x, PRA) verifies if there is enough water while the main crop's growing season to grow a second crop with the resting resources. ASSESS_Nutrients_Companion Crop(x, PRA) works the same but for soil nutrients. If resources are judged insufficient by one of these both functions, the assessed crop is deleted from the list of eligible companion crop. If the list is empty after these both functions, there is no companion crop.

At the end of the function, the last entry of x.rotat[PRA] is updated by adding the selected companion crop. If there is no one, x.SelectedCC = None.

3.2.7. "APPLY" functions: Harvesting and decomposition

The harvest is simulated in the function APPLY_SelectedCrop_Harvest (PRA, x). This time, the nutrients amounts from the dictionary removed is subtracted to x.ActualStand – see 3.2.3) ASSESS_Nutrients(x, PRA): Soil nutrients availability. The cut of the companion crop works the same way and is simulated in the function APPLY_SelectedCC_Kill(PRA,x).

The functions APPLY_ResiduesDecomposition_of_SelectedCrop(x) and APPLY_ResiduesDecomposition_of_CompanionCrop(x) respectively ensure the simulation of the residues decomposition from the main crop and its companion crop. For the next 90 months after the death of the crop, the released nutrients amount is computed according to the function of organic carbon decomposition from the model STICS[1,2] – here, the function mineralizedCPK(crop, month). As the nitrogen mineralization function – mineralizedN(crop, month) – does still not work as shown in the article of Justes et al. (2009)[3], nitrogen amount is calculated by dividing the mineralized carbon amount with the residues' C:N ratio. The function APPLY_ResiduesDecomposition_of_PreviousCrops(PRA, x) adds the value x.decomposition_month[1][nutrient] to x.ActualStand[PRA][nutrient] and switch from one month all values from x.decomposition_month for each month of the x.SelectedCrop's growing season. The keys of this dictionary represent the amount of months before the nutrient's mineralization.

3.3. Third step: assessing the feasibility of self-reliance according to the average nutritional value of harvested products.

The last step of the program is a function called `MDL_QTTperPERSON`. It runs in two times.

Firstly, it calculates from the total yields (‘`x.totalYields`‘) the average weekly resources for each crop in amount of pieces. That way, it becomes easier to estimate the harvested quantities. Then, it sums the amount of nutrients and vitamins of all products in the corresponding entry of the dict '`x.TotalNutrients`' . Each key corresponds to a nutrient, a vitamin or another dietary feature.

In a second time, new entries are created in the dict ‘`x.dietary_results`’ by dividing the amount of each dietary feature by the required intake of total population. This process takes the population pyramid of 2017 into account. The required intake of total population is calculated by multiplying the population’s dietary requirements of both gender for each age by the related population and summed to get the total needs of the French population in 2017.

That way, at the end of the function, ‘`x.dietary_results`’ contains the average nutrient quantity per person

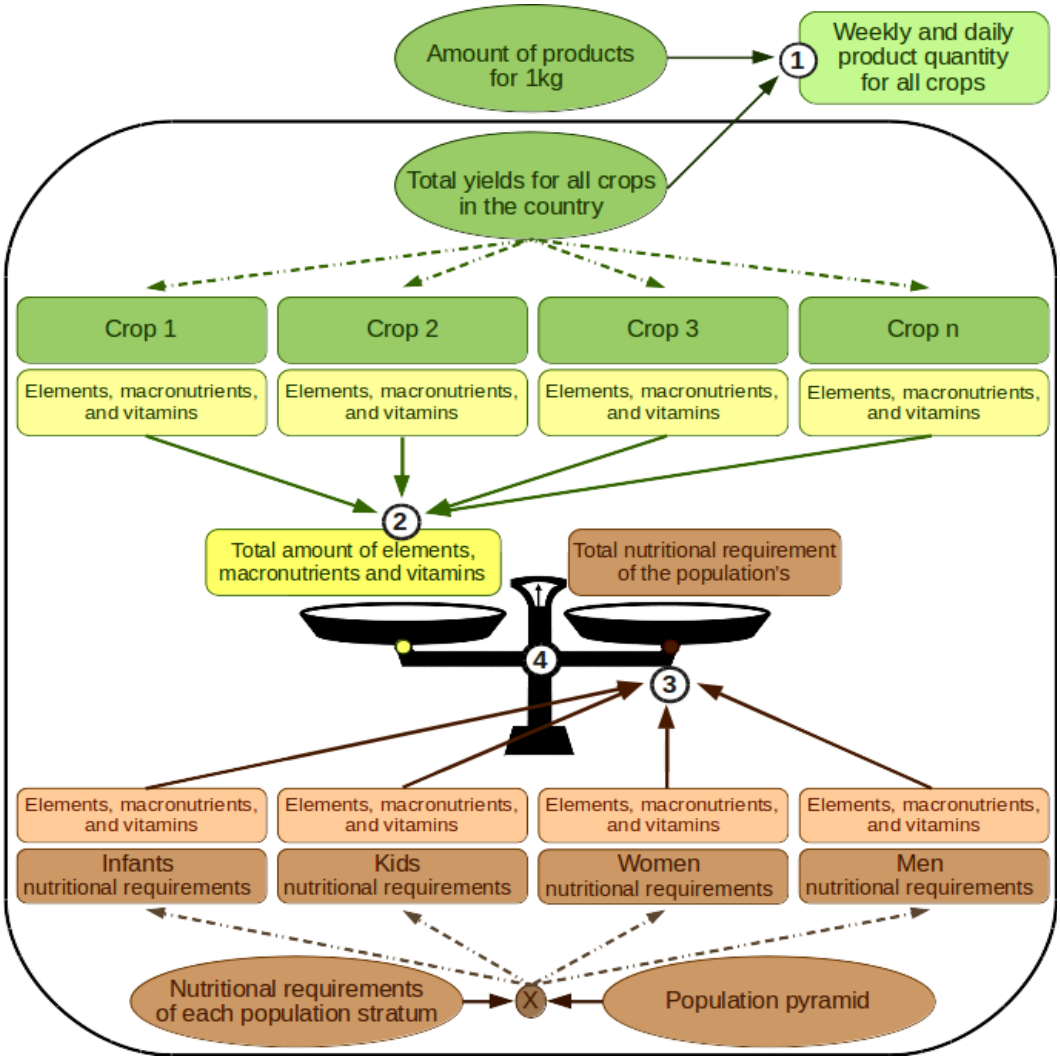

Figure 3: Schematic representation of VegAu's 'step 3': ① Calculating the weekly and daily amount of products, ② Summing all nutritional features of the simulated yields, ③ Summarizing the total nutritional requirement of the whole population.

as well as the proportion it represents compared to the three thresholds given by Canada Health: the Estimated Average Requirements, the Recommended Dietary Allowances (or Adequate Intakes) and the Tolerable Upper Intake Levels.

4. Key Variables: detailed approach

4.1. Introduction to the VegAu's key variables

The variables called "key variables" in this Master thesis are variables that are used all along the model, or which need to be used and modified by several functions. Some of them also contain the results of a certain part of the model.

The steps 1 and 3 are relatively short and don't use a lot of special variables. The step 2, however, is composed by lots of different variables which have been separated in four different categories according to their role in the simulation process: coordination variables, temporal variables, selection variables and analytical variables. At least, a single variable called "results" is common to the steps 2 and 3 and gathers the main results of these both steps.

The called "key values" are all part of the class "x", so they appear as "x.VariableName" in the code. In Python, affecting variables to a class allow them to be modified by the functions while remaining available after the process: else, function's variables are destroyed when it ends.

Before to begin with these explanation, it is important to define some specific words. Variables called "dictionaries", associate a key word or expression to a list, a number or another word or dict. In VegAu, the dictionaries' key words (named "key") are often the ID of the areas of interest, of the crops or both. "Boolean" are values that can be set as `True` or `False`; they are principally the condition for an assessment to continue or to be stopped.

All VegAu's most important variables are respectively referenced in the Appendix 1, 2 and 3 for the step 1, 2 and 3 with the list of all functions they are part of. The Appendix 4, at the contrary, lists all functions and associates their related variables to them.

4.2. Variables from the step 1

The boolean `all_crop_parameters_match_the_PRA_ones` masters the main loop of the step 1. It is set to `True` by default for each crop of the database for each PRA. If the current crop fails to an eligibility test[a], `all_crop_parameters_match_the_PRA_ones` becomes `False`: the loop is broken and the next crop is evaluated.

The first eligibility test concerns the minimum temperature requirement. The list `eligible_Tmin` contains as much entry as the amount of months for the current crop's longest growing season. For each month, a boolean is added to the list: if the PRA's minimum temperature is higher than the minimum temperature supported by the crop, the month is "`True`", and if it is too cold, it is "`False`". If there is at least as much "`True`" consecutive months as the shortest growing season, the crop stays eligible and can pass to the next test. `eligible_Tmin` is also reused in the water assessment function of the first step (`ASSESS_Water`).

a Minimum temperature, sunlight requirement, water requirements, pH...

The water resources quality assessment does not use a variable from the class “x” to count eligible months, while the indices `TOLERdrought` and `TOLERflood` do. The latter are used to estimate the water resources quality and have to be modified by the function `CORR_TOLERdf` in order to change the original database index (from 1 to 10) into a percentage.

The last special variables are the dictionaries `eligibleCropsEN`, `eligibleCropsFR`, `eligibleCropsDE` and `eligibleCropsID`. They respectively associate each PRA to their list of eligible crops in French, English and German. The crop’s IDs (`eligibleCropsID`) are used in the next steps while the common names in French, English and German have been created for a clearer graphic interface.

4.3. Variables from the step 2

4.3.1. Coordination variables

Variables called “coordination variables” are booleans or dictionaries. They allow to stop the main loops if there no eligible crop remain for the current season or if the limiting factor has been reached. That way, `LimitingFactorReached` is set to `False` before the rotation simulation of each PRA and `no_delay_because_of_T_or_water` is set as `True` before the temperature and water assessments. If there are not enough water resources or if the season is too cold for the selected crops, `no_delay_because_of_T_or_water` becomes `False`, the earlier seeding date is delayed and the tests go further with the next crops.

`decomposition_month` is a dict. It masters the amount of nutrients that are returned to the soil after the crop’s decomposition, up to eight years after their harvest. Each key corresponds to a month following the crop’s death and is associated to the amount of N, P, K and OM that should be released after this delay.

4.3.2. Temporal variables

The “temporal variables” corresponds to the duration (in months) of the rotation until a specific event. For instance, `GSstart` gives the earlier planting date of a crop after the earlier harvesting date of the previous crop (`EndPreviousCrop_earlier`). Until no crop has been selected to follow the previous one, it is a dict containing an entry for each eligible crop. Once a crop has been selected, `GSstart` becomes the planting month (int) of the selected crop.

`EndPreviousCrop_earlier` and `EndPreviousCrop_later` are respectively the earlier and later harvesting date of the last selected crop. When a rotation ends because of the lack of nutrients, `EndPreviousCrop_later` corresponds to the total rotation duration in months.

4.3.3. Selection variables

These variables manage the crop’s selection all along the model. In particular, `eligibleCrops` corresponds to the PRA’s `eligibleCropsID` list and is cleaned or updated according to each eligibility test, from the seeding date assessment until the selection of a crop.

`eligibleCompanionCrops` is the list of eligible crops that can be grown with the selected crop in order to cover the ground while bringing nitrogen to the soil, just like clover. It is updated in the function `SELECT_CashCrop` and reused in `SELECT_CompanionCrop`.

`LaterCrops` is used if the `eligibleCrops` list becomes empty before the selection of a crop. It contains the next crops with their earlier seeding date.

Once all tests have been executed, several indexes from 0 to 1 are used to determine the best crop to select. They have been calculated while each preceding test and are then crossed to get a final selection index: `indexWR`, `indexPnD`, `indexNutrients` and `indexDelay` concern respectively the Water Resources, the Pests and Diseases, the Nutrients Margins and an eventual delay after the later harvesting date of the previous crop.

`CCeligibility` is a dict used in `SELECT_CompanionCrop`. It contains the water resources indexes that have been calculation in `ASSESS_Water_CompanionCrop` for all eligible companion crops.

Finally, once a crop is selected, the variables `SelectedCrop`, `SelectedCC` and `PreviouslySelectedCrop` are updated. They are used all along the rotation simulation as references for the selection of the next crops.

4.3.4. Analytical variables

Following variables help to analyse:

- the territorial repartition of crops and nutrients,
- limiting factors of rotations,
- yields
- the construction of rotations.

`prodSURFACE` is calculated in the first step but is indirectly used in the second one for the crop selection. It gives a nice overview of the adaptation capacity, the potential availability of a crop at the country scale. `ActualStand` gives the nutrients amount in the soil for each month of the rotation. At the end of the simulation, its values corresponds to the final stand of the soil at the end of the rotation.

Limiting Factors, for instance the lack of nutrients or pests and diseases, are managed by four variables. `LimitingFactor` is a dict which associates the limiting nutrient(s) with the PRA ID. If `eligibleCrops` is empty because of the lack of nutrients, the limiting factor has been reached: a last entry is added to `rotat` with the rotation duration is associated to the PRA ID in the `rotation_length` dict and the list of all limiting nutrients. Pests and diseases are firstly managed thanks to the dict `VERIFprodBOT` to count the months between two crops of a same botanical family as well as the total amount of crop of a same botanical family in the rotation. `PestsDiseases_in_rotation` is not used in the simulation: it is an index that count how much times a crop has been selected while it could be subject to pests and diseases. It allows to get an overview of the effectiveness of the rotation's health.

Yields are managed by the dict `totalYields`: for each PRA, it attributes a yield estimation to each crop of the rotation according to the water resources quality and the pests and diseases risks. At the end of the rotation, the yields of a PRA are divided by the rotation length in order to get an average yearly value. When all PRA have been assessed, `totalYields` is updated to only contain the average yearly yield of all crops in whole France. These values are then used in the step 3.

Special variables have been set for rotations analysis, in order to verify the feasibility of rotations. `rotat` counts how much (different) crop succeeded while the rotation in how much time and to know how much PRA grew a permanent crop. `CHOICE` saves the selection level of each crop of the rotation: unused at the local and national scale, unused at the local scale, already used cash crop, cover crop… Finally,

`representativity` is a dict which associates the ID of each selected crop to its occurrence in rotation all over the country.

4.4. Variables from the step 3

`TotalNutrients` contains the percentage of the minimum, maximum and average (daily) intake amount of each nutrient for the considered population.

`DailyResources` associates the ID of each crop to the average weekly amount of fruits (pieces), vegetable (pieces) or flour (weight) which could be available for each person.

5. Common and final variable: ‘results’

`results` is a dict which summarize all informations that are meant to be useful by the evaluation of the model and its results. For each PRA, the concerned informations are the following:

1) the rotation length,
2) the amount of different crops in the rotation,
3) the total amount of crops in the rotation,
4) the ID of different crops in the rotation,
5) the ID of all crops in the rotation,
6) the permanent crop amount (theoretically, 1 or 0),
7) the permanent crop ID if there is one,
8) the limiting Factor(s)

6. APPENDIX

Appendix 1: Summary of VegAu's key variables for the step 1 and those which appear in both steps 1 and 2.

Technically, the so-called “key variables” are "self variables" from the class x: that is why their names are preceded by “x.”. Used variable types are: int (int), float (float), list (list), dict (dict), and booleans (bool).

Related functions are in bold if they are primary ones and in italic if they are secondary ones.

Complete name	Type	Refers to	Related function(s)
STEP 1			
`x.all_crop_parameters_match_the_PRA_ones`	bool	Main loop	**`MDL_eligibilityTest`**`(x,data)` *`ASSESS_Tmin`*`(crop,PRA,x)` *`ASSESS_Tmin_germ_forFruits`*`(crop,PRA,x)` *`ASSESS_Sunshine`*`(crop,PRA,x)` *`ASSESS_Water`*`(crop,PRA,x)` *`ASSESS_pH`*`(crop,PRA,x)`
`x.eligible_Tmin`	list	crops	*`ASSESS_Tmin_germ_forFruits`*`(x,crop,PRA)` *`ASSESS_Tmin`*`(crop,x,PRA)` *`ASSESS_Water`*`(crop,PRA,x)`
`x.eligibleCropsEN`	list	Crops in PRA	**`MDL_eligibilityTest`**`(x, data)`
`x.eligibleCropsFR`	list	Crops in PRA	**`MDL_eligibilityTest`**`(x, data)`
`x.eligibleCropsDE`	list	Crops in PRA	**`MDL_eligibilityTest`**`(x, data)`
COMMON VARIABLES (STEP 1 AND 2)			

`x.TOLERdrought`	float	crops	**`STEP 1`**`----------`
			`CORR_TOLERdf(crop, x)`
			`ASSESS_Water``(crop, PRA, x)`
			`STEP 2``----------`
			`WRmargin_GSmin(crop, month, x, PRA)`
			`ASSESS_WaterResources``(PRA, x)`
			`SELECT_CashCrop``(x, PRA, data)`

`x.TOLERflood`	float	crops	**`STEP 1`**`----------` `CORR_TOLERdf(crop, x)` *`ASSESS_Water`*`(crop, PRA, x)` **`STEP 2`**`----------` *`ASSESS_WaterResources`*`(PRA, x)`
`x.eligibleCropsID`	list	Crops in PRA	**`STEP 1`**`----------` **`MDL_eligibilityTest`**`(x, data)` **`STEP 2`**`----------` **`MDL_Rotation`**`(x, data)` *`ASSESS_SeedingDate`*`(PRA, x)`

Appendix 2: Summary of VegAu's key variables for the step 2.

Technically, the so-called “key variables” are "self variables" from the class x: that is why their names are preceded by “x.”. Used variable types are: int (int), float (float), list (list), dict (dict), and booleans (bool). Related functions are in bold if they are primary ones and in italic if they are secondary ones.

Complete name	Type	Refers to	Related function(s)
Coordination Variables			
x.LimitingFactorReached	bool	Main loop	**MDL_Rotation**(x, data) *ASSESS_SeedingDate*(PRA, x) VERIF_lastCrops_not_CC(x,PRA,nutrient) *ASSESS_NutrientsMargin*(PRA, x) *ASSESS_Nutrients*(x, PRA)
x.no_delay_because_of_T_or_water	bool	Main loop	**MDL_Rotation**(x, data) *ASSESS_WaterResources*(PRA, x)
x.decomposition_month	dict	Soil (PRA)	**MDL_Rotation**(x, data) ASSESS_NutrientsMargin(PRA, x) ASSESS_Nutrients_CompanionCrop(x, PRA) *APPLY_ResiduesDecomposition_of_PreviousCrops*(PRA, x) APPLY_ResiduesDecomposition_of_CompanionCrop(x) APPLY_ResiduesDecomposition_of_SelectedCrop(x) mineralizedN(crop, month) mineralizedCPK(crop, month)
Temporal Variables			
x.GSstart	dict and int	eligible crops and selected crop	**MDL_Rotation**(x, data) WRmargin_GSmin(crop, month, x, PRA) ASSESS_SeedingDate(PRA, x) ASSESS_WaterResources(PRA, x) SELECT_CashCrop(x, PRA, data) APPLY_ResiduesDecomposition_of_PreviousCrops(PRA, x) UPDATE_VERIFprodBOT_and_PestsDiseases_in_rotation(PRA, x) **STEP 1----------** WaterResources(month,Gsstart,PRA,crop,x)
x.EndPreviousCrop_earlier	int	Previously selected crop (PRA, main	**MDL_Rotation**(x, data) ETc_GSmax(crop, month, x, PRA) ETc_GSmin(crop, month, x, PRA) ASSESS_SeedingDate(PRA, x)

Complete name	Type	Refers to	Related function(s)
		loop)	ASSESS_WaterResources(PRA, x) SELECT_CashCrop(x, PRA, data) UPDATE_VERIFprodBOT_and_PestsDiseases_in_rotation(PRA, x)
x.EndPreviousCrop_later	int	Previously selected crop (PRA, main loop)	**MDL_Rotation**(x, data) *ASSESS_SeedingDate*(PRA, x) *ASSESS_WaterResources*(PRA, x) ASSESS_NutrientsMargin(PRA, x) *SELECT_CashCrop*(x, PRA, data) *UPDATE_VERIFprodBOT_and_PestsDiseases_in_rotation*(PRA, x)
Selection Variables			
x.eligibleCrops	list	Step 1, eligible crops IDs (PRA)	**MDL_Rotation**(x, data) *VERIF_TreesInRegion*(PRA, x, data) *ASSESS_SeedingDate*(PRA, x) *ASSESS_WaterResources*(PRA, x) VERIF_lastCrops_not_CC(x, PRA, ASSESS_NutrientsMargin(PRA, x) *ASSESS_Nutrients*(x, PRA) *ASSESS_PestDiseases*(x, PRA) *SELECT_CashCrop*(x, PRA, data)
x.eligibleCompanionCrops	list	Step 2, updated eligible crops list	**MDL_Rotation**(x, data) *SELECT_CashCrop*(x, PRA, data) ASSESS_Water_CompanionCrop(x, PRA) ASSESS_Nutrients_CompanionCrop(x,PRA) *SELECT_CompanionCrop*(x, PRA)
x.indexWR	dict and float	eligible crops and Selected Crop	*VERIF_TreesInRegion*(PRA, x, data) *SELECT_CashCrop*(x, PRA, data) ASSESS_Nutrients_CompanionCrop(x, PRA) **MDL_Rotation**(x, data)
x.indexPnD	dict and float	eligible crops and Selected Crop	*ASSESS_PestDiseases*(x, PRA) *SELECT_CashCrop*(x, PRA, data) **MDL_Rotation**(x, data)
x.indexNutrients	dict and float	eligible crops and Selected Crop	ASSESS_NutrientsMargin(PRA, x) *ASSESS_Nutrients*(x, PRA) ASSESS_Nutrients_CompanionCrop(x, PRA) *SELECT_CompanionCrop*(x, PRA) **MDL_Rotation**(x, data)
x.indexDelay	dict	eligible crops	*ASSESS_SeedingDate*(PRA, x)

Complete name	Type	Refers to	Related function(s)
	and float	and Selected Crop	*SELECT_CashCrop*(x, PRA, data)
x.SelectedCrop	str	Plants DB (ID)	*VERIF_TreesInRegion*(PRA, x, data) *SELECT_CashCrop*(x, PRA, data) ASSESS_Water_CompanionCrop(x, PRA) ASSESS_Nutrients_CompanionCrop(x, PRA) *SELECT_CompanionCrop*(x, PRA) *APPLY_ResiduesDecomposition_of_PreviousCrops*(PRA, x) APPLY_ResiduesDecomposition_of_SelectedCrop(x) *APPLY_SelectedCrop_Harvest*(PRA, x)
x.SelectedCC	str	Plants DB (ID)	*SELECT_CashCrop*(x, PRA, data) ASSESS_Water_CompanionCrop(x, PRA) ASSESS_Nutrients_CompanionCrop(x, PRA) *SELECT_CompanionCrop*(x, PRA) APPLY_ResiduesDecomposition_of_CompanionCrop(x) *APPLY_SelectedCC_Kill*(PRA, x)
x.PreviouslySelectedCrop	str	Plants DB (ID)	**MDL_Rotation**(x, data) *SELECT_CashCrop*(x, PRA, data)
Analytical Variables			
x.prodSURFACE	dict	Step 1	**MDL_eligibilityTest**(x, data) *ASSESS_Priority*(x, data) **MDL_Rotation**(x, data)
x.ActualStand	dict	Soil (PRA)	*ASSESS_NutrientsMargin*(PRA, x) *ASSESS_Nutrients_CompanionCrop*(x, PRA) *APPLY_ResiduesDecomposition_of_PreviousCrops*(PRA, x) *APPLY_SelectedCC_Kill*(PRA, x) *APPLY_SelectedCrop_Harvest*(PRA, x) OM_retention_capacity(x, PRA) AWC_SoilOnly(PRA) **MDL_Rotation**(x, data)
x.LimitingFactor	dict	PRA (results)	**MDL_Rotation**(x, data) *ASSESS_NutrientsMargin*(PRA, x)
x.rotation_length	dict	PRA (results)	**MDL_Rotation**(x, data)
x.VERIFprodBOT	dict	PRA, bo tanic families	**MDL_Rotation**(x, data) *ASSESS_PestDiseases*(x, PRA) *UPDATE_VERIFprodBOT_and_PestsDiseases_in_rotation*(PRA, x)
x.PestsDiseases_in_rotation	dict	PRA	*UPDATE_VERIFprodBOT_and_PestsDiseases_*

Complete name	Type	Refers to	Related function(s)
		(index)	*in_rotation*(PRA, x)
x.totalYields	dict	Results (PRA and total)	**MDL_Rotation**(x, data)
x.rotat*	dict	Results (PRA)	*ASSESS_SeedingDate*(PRA, x) *VERIF_TreesInRegion*(PRA, x, data) *VERIF_lastCrops_not_CC*(x, PRA, nutrient) *ASSESS_PestDiseases*(x, PRA) *SELECT_CashCrop*(x, PRA, data) *SELECT_CompanionCrop*(x, PRA)
x.CHOICE	dict	Reculs (PRA)	**MDL_Rotation**(x, data) *VERIF_TreesInRegion*(PRA, x, data) *SELECT_CashCrop*(x, PRA, data)
x.representativity	dict	Results (crops ID)	*VERIF_TreesInRegion*(PRA, x, data) *SELECT_CashCrop*(x, PRA, data)

** Notice : x.rotat is a dictionary containing lists of tuples for each spatial unit. These tuples are composed that way:*

(x.SelectedCrop, x.SelectedCC, x.rotat[PRA][-1][2], x.EndPreviousCrop_later)

Appendix 3: Summary of VegAu's key variables for the step 3.

Technically, the so-called “key variables” are "self variables" from the class x: that is why their names are preceded by “x.”. Used variable types are: int (int), float (float), list (list), dict (dict), and booleans (bool).

Related functions are in bold if they are primary ones and in italic if they are secondary ones.

Complete name	Type	Refers to	Related function(s)
`x.TotalNutrients`	dict	Crops used in the step 2	**`MDL_QTTperPerson`**`(x, nutrition)`
`x.WeeklyyResources`	dict	Crops used in the step 2	**`MDL_QTTperPerson`**`(x, nutrition)`
MAIN OUTPUT (summarize the main results from the three steps)			
`x.results`	dict	Contains: • rotation length • amount of different crops in the rotation • total amount of crops in the rotation • ID of different crops in the rotation • ID of all crops in the rotation • permanent crop amount (theoretically, 1 or 0) • the permanent crop ID if there is one • limiting Factor(s)	

Appendix 4: Summary of VegAu's main functions with their most important variables.

Used variable types are: int (int), float (float), list (list), dict (dict), and booleans (bool).

Step	Name	Role	Variables	Type

1	MDL_eligibilityTest(x, data)	**Primary**	`x.eligibleCropsID[PRA]`
			`x.eligibleCropsEN[PRA]`
			`x.eligibleCropsFR[PRA]`
			`x.eligibleCropsDE[PRA]`
			`x.all_crop_parameters_match_the_PRA_ones`
				`list`
				`list`
				`list`
				`list`
				`bool`

1	ASSESS_Tmin(crop,x,PRA)	*Secondary*	`x.eligible_Tmin`
			`the_current_crop_is_a_permanent_crop`
			`PRAedibTest`
			`Tmin_eligibility`
			`x.all_crop_parameters_match_the_PRA_ones`
				`list`
				`bool`
				`list`
				`bool`
				`bool`

1	ASSESS_Tmin_germ(crop,x,PRA)	*Secondary*	`x.eligible_Tmin_germ`
			`x.eligible_Tmin`
			`edibTest`
			`x.all_crop_parameters_match_the_PRA_ones`
				`list`
				`list`
				`int`
				`bool`

1	ASSESS_Sunshine(crop,x,PRA)	*Secondary*	`eligible_Sunshine`
			`crop_needs_sun`
			`enough_sunshine`
			`x.all_crop_parameters_match_the_PRA_ones`
				`list`
				`bool`
				`bool`
				`bool`

1	ASSESS_Water(crop,x,PRA)	*Secondary*	`eligible_WaterRqt`
			Kc1_4(crop), Kc2_4(crop), Kc3_4(crop), Kc4_4(crop)
			ETPmoy(month, PRA)
			`x.all_crop_parameters_match_the_PRA_ones`
				`list`
				`float`
				`lambdas`
				`lambda`
				`bool`

1	ASSESS_pH(crop,x,PRA)	*Secondary*	pH(PRA) `x.all_crop_parameters_match_the_PRA_ones`	`lambda` `bool`

1	ASSESS_Priority(x,PRA)	*Secondary*	`data.environment`
			`data.plants`
			`x.prodSURFACE`
			`QttPerInhabitant`
			`ratioADAPT`
			prodCAT(crop)
				`dict`
				`dict`
				`dict`
				`float`
				`float`
				`lambda`

Step	Name	Role	Variables	Type
2	MDL_Rotations(crop,x,PRA) (these variables are reset for each PRA)	**Primary**	`x.CHOICE`	`dict`
			`x.PreviouslySelectedCrop`	`None`
			`x.SelectedCrop`	`None`
			`x.SelectedCC`	`None`
			`x.EndPreviousCrop_earlier`	`int`
			`x.EndPreviousCrop_later`	`int`
			`x.decomposition_month`	`dict`
			`x.eligibleCrops`	`list`
			`x.eligibleCompanionCrops`	`list`
			`x.rotat`	`dict`
			`x.indexWR`	`dict`
			`x.indexPnD`	`dict`
			`x.indexNutrients`	`dict`
			`x.LimitingFactor`	`dict`
			`x.VERIFprodBOT`	`dict`
			`x.PestsDiseases_in_rotation`	`dict`
			`x.ActualStand`	`dict`
			`x.totalYields`	`dict`
			`x.LimitingFactorReached`	`bool`
			`x.no_delay_because_of_T_or_water`	`bool`
			`the_selected_crop_is_a_permanent_crop`	`bool`
			`x.decomposition_month`	`dict`
2	`ASSESS_SeedingDate(PRA, x)`	*Secondary*	`x.eligibleCrops_init`	`list`

Population category	Age bracket	
Male and female infants	0 to 6	months
	7 to 12	months
Male and female children	1 to 3	years
	4 to 8	years
Males and Females (teenagers and adults)	9 to 13	years
	14 to 18	years
	19 to 30	years
	31 to 50	years
	51 to 70	years
	< 70	years

Table 3: Selected population categories and age brackets in the Canada Health Dietary Intakes Tables

Step	Name	Role	Variables	Type
			x.eligibleCrops	list
			x.rotat	int
			x.EndPreviousCrop_earlier	int
			x.EndPreviousCrop_later	dict
			x.indexDelay	dict
			x.GSstart	list
			x.laterCrops	list
2	ASSESS_WaterResources(PRA,x)	*Secondary*	x.TOLERdrought	int
			x.TOLERflood	int
			x.indexWR	dict
			x.eligibleCrops	list
			CORR_TOLERdf(crop, x)	function
			x.GSstart	dict
			x.EndPreviousCrop_earlier	int
			x.EndPreviousCrop_later	int
			x.rotat	dict
			x.no_delay_because_of_T_or_water	bool
2	ASSESS_Nutrients(x, PRA)	*Secondary*	x.LimitingFactorReached	Bool
			x.indexNutrients	dict
2	ASSESS_NutrientsMargin (PRA, x)	Tertiary	x.eligibleCrops	dict
			x.indexNutrients	dict
			removed	dict
			x.ActualStand	dict
			x.LimitingFactor	dict
			x.LimitingFactor_crops	dict
			x.LimitingFactorReached	bool
2	ASSESS_PestDiseases(x, PRA)	*Secondary*	x.indexPnD	dict
			x.VERIFprodBOT	dict
			x.eligibleCrops	list
2	SELECT_CashCrop(x,PRA,data)	*Secondary*	x.CHOICE	dict
			x.eligibleCrops	list
			x.eligibleCompanionCrops	list
			eligibleCoverCrops	list
			eligibleCashCrops	list
			unusedCrops	list
			unusedCrops_countryScale	list
			unusedCrops_without_delay	list
			unusedCrops_countryScale_delay	list
			unusedCashCrops	list

Step	Name	Role	Variables	Type
			x.GSstart	**dict → int**
			x.SelectedCrop	str
			Final_eligibility_Index	dict
			SelectionDone	bool
2	SELECT_CompanionCrop(x, PRA)	*Secondary*	x.SelectedCC	str
			x.eligibleCompanionCrops	list
			x.rotat	dict
2	ASSESS_Water_CompanionCrop (x, PRA)	*Tertiary*	x.SelectedCC	str
			x.eligibleCompanionCrops	list
			Kc1_4(crop), Kc2_4(crop), Kc3_4(crop), Kc4_4(crop)	lambdas
			ETPmoy(month, PRA)	lambda
2	ASSESS_Nutrients_Companion Crop(x, PRA)	Tertiary	x.SelectedCC	str
			x.eligibleCompanionCrops	list
			removed	dict
			x.indexNutrients	dict
			x.ActualStand	dict
2	APPLY_ResiduesDecomposition _of_PreviousCrops(PRA, x)	*Secondary*	x.ActualStand	dict
			x.decomposition_month	dict
2	APPLY_SelectedCC_Kill(PRA,x)	*Secondary*	x.SelectedCC	str
			removed	dict
			x.ActualStand	dict
2	APPLY_ResiduesDecomposition _of_CompanionCrop(x)	Tertiary	Residues	dict
			mineralizedN(crop, month)	function
			mineralizedCPK(crop, month)	fonction
			x.decomposition_month	dict
2	APPLY_SelectedCrop_Harvest (PRA, x)	*Secondary*	removed	dict
			x.ActualStand	dict
2	APPLY_ResiduesDecomposition _of_SelectedCrop(x)	Tertiary	Residues	dict
			mineralizedN(crop, month)	function
			mineralizedCPK(crop, month)	function
			x.decomposition_month	dict
3	MDL_QTTperPerson (x,nutrition)	**Primary**	x.dietary_results	dict
			x.TotalNutrients	dict
			x.totalYields	dict
			x.WeeklyResources	dict
			totalPopulation	int
			CropNutrients	dict

Step	Name	Role	Variables	Type
			`Canada_Health`	`dict`
			`PopulationPyramid`	`dict`
			`x.dietary_results`	`dict`

7. LITERATURE

(1) Nadine Brisson, « Questics : le STICS, c'est quoi ? », popularization and documantation article about the STICS model (estimated to date from 2000)

(2) STICS official Website, « Présentation du modèle STICS » : http://www6.paca.inra.fr/stics/Qui-sommes-nous/Presentation-du-modele-Stics

(3) E. Justes, B. Mary, B. Nicolardot (2009). *Quantifying and modelling C and N mineralization kinetics of catch crop residues in soil: parameterization of the residue decomposition module of STICS model for mature and non mature residues*, Springer Science and Business Media B.V. 2009, Plant and Soil – December 2009, DOI 10.1007/s11104-009-9966-4